大語彙連続音声認識エンジン Julius ver. 4

李 晃伸

平成 19 年 12 月 18 日

1 はじめに

大語彙連続音声認識 Julius の ver. 4.0（以下 Julius-4 と表記）は，大幅なソースの更新や再構成を含む意欲的な更新となっている．大きな変更点として，(1) エンジン内部構造の整理・ライブラリ化，および複数モデルを組み合わせたマルチデコーディングの実現，(2) 言語制約の拡張（任意長 N-gram，ユーザ定義言語制約関数，文法との統合，単単語認識の追加など），(3) 雑音環境下での音声区間検出の強化，が行われた．これらの発展は，今回行われた Julius の内部構造のモジュール化およびソースの全面的な再構成により，可搬性と柔軟性が大幅に向上された結果実現されたものである．また，メモリ効率等の性能改善も行われた．

使い方は，従来のバージョンと互換性を維持している．また，ver.4.0 では認識精度および処理速度は従来から変化していない．これについては次期リリース以降で改善する予定である．

以下，Julius 4.0 の構成について解説した後，Julius 4.0 での変更点や新機能について解説する．

2 Julius-4 の構成

2.1 内部構成

Julius-4 の内部構成を図 1 に示す．認識エンジン全体はひとつの「エンジンインスタンス」として定義されており，内部は大きく分けて，認識実行用の各種インスタンスおよび設定パラメータ群からなる.「音響モデルインスタンス」は，音響 HMM および音響尤度計算用のワークエリア・キャッシュを持つ.「言語モデルインスタンス」は，単語 N-gram，文法，あるいは孤立単語認識用辞書といった言語モデルのタイプに分かれ，それぞれモデル情報および言語計算のためのワークエリアを持つ.「認識処理インスタンス」は認識を実行する単位であり，認識に必要な木構造化辞書や探索用ワークエリア等を持つ．この認識インスタンスは，使用する音響モデルおよび言語モデルのインスタンスをそれぞれ参照する．また言語モデルは，辞書読み込みのために，対応する音響モデルのインスタンスを参照する．

図中右上の「設定パラメータ」は，様々な認識用パラメータの設定値を格納している．Julius-4 では複数のモデルを用いることもできるが，その場合それぞれの設定情報は AM 設定，LM 設定，認識器設定といった構造体ごとにリスト管理される．

起動時，Julius-4 はまず，コマンド引数や設定ファイル (jconf ファイル) からパラメータを読み込む．複数のモデルが指定されている場合は，それぞれに対して設定構造体を定義する．その後，与えられた設定パラメータ内の各構造体に従って，対応する各種インスタンスを生成する．その後，各インスタンスはモデルを読み込み，認識の準備を行う．

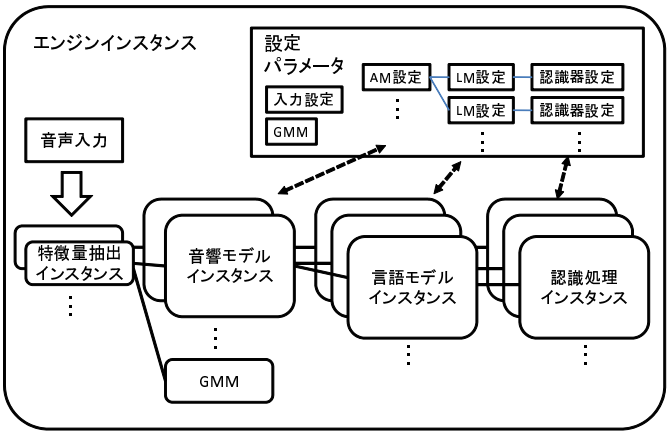


図 1: Julius ver.4.0 内部構成

特徴量の設定は独立しておらず，音響モデル設定に含まれる．Julius-4 の起動時には，音響モデルインスタンスごとに必要とされる特徴量の型が（音響モデルヘッダ等から）決定され，それらに対応する特徴量（MFCC）インスタンスが内部生成される．特徴量インスタンスは，特徴量の計算のほか，リアルタイム CMN やスペクトルサブトラクションも行う．同じ特徴量タイプを持つ複数の音響モデルは，ひとつの特徴量インスタンスを共有する．なお，GMM についても同様である．

Julius-4 では音声入力ソースは単一のみ扱う．

2.2 認識処理

認識実行時の動作は基本的にこれまでの Julius と同様である．複数の認識処理インスタンスが定義されている場合は，第 1 パスではそれぞれの認識処理がフレーム同期で並列に進行され，その後第 2 パスが順次実行される．

計算時のキャッシュやワークエリアは，それぞれの音響モデルインスタンスや言語モデルインスタンス内に保持されている．このため，たとえば一つの音響モデルを複数の言語モデルで共有している場合などでは，キャッシュを共有しているため単純にエンジンを並べた並列認識よりも効率よく計算できる．

3 内部構造の整理とライブラリ化

3.1 内部構造の整理

- エンジンの内部構造の整理：前節で述べたように，エンジンの内部構造を全体的に見直して再構成した．
- 全グローバル変数を整理して階層構造化：従来多用されていたグローバル変数を，すべて構造体として定義しなした．また，階層的に構造化を行い，整理した．

これらの改善により，後述のライブラリ化や複数モデルのマルチデコーディング，言語制約の拡張などが実現できた．

3.2 ライブラリ化

- エンジンコア部（認識実行部）のライブラリ化：アプリケーションとしての Julius のみを `julius` に残し，大部分のエンジンコア部はライブラリとしてディレクトリ `libjulius` に分離された．アプリ側に分離された機能は，文字コード変換，音声録音，モジュールモードの 3 機能である．
- コールバックの実装：認識処理の内容や結果をアプリケーションが知る方法として，コールバックを指定しておけるようになった．認識の結果のみならず，様々なタイミングでユーザ定義関数を呼び出すことができるので，アプリケーションとより緊密な連携を行える．
- API の整備：アプリから任意のタイミングで認識を中断・再開，文法の追加や削除をできるよう関数を整備した．また，任意の言語モデルや認識処理インスタンスを Julius-4 の実行中に動的に追加・削除できるようになった．
- 出力先の整備：ライブラリの全てのテキスト出力を関数 `jlog()` に一本化した．オプションの指定あるいは関数呼び出しによって，出力先をファイルに変更したり，あるいはまったく出力しないようにできる．

3.3 コンパイル時オプションの統合

従来の Julius は，Julian を含め，コンパイル時に指定して別実行ファイルとするオプションが多く存在した．Julius-4 ではこれらのコードの統合を進めた．

まず，複数の言語モデル（N-gram, 文法，孤立単語）を同時サポートした．これに伴い Julian は Julius に統合された．Julius-4 に文法を与えることで旧 Julian と同じ動作となる．

次に，単語グラフ出力およびショートポーズセグメンテーションを通常オプション化した (`-lattice` および `-spsegment`)．

マルチパス版については，音響モデルのタイプをチェックして，必要であれば自動的にマルチパス版モードに移行するようになった．明示的にマルチパスモードで動作させることもできる (`-multipath`)．

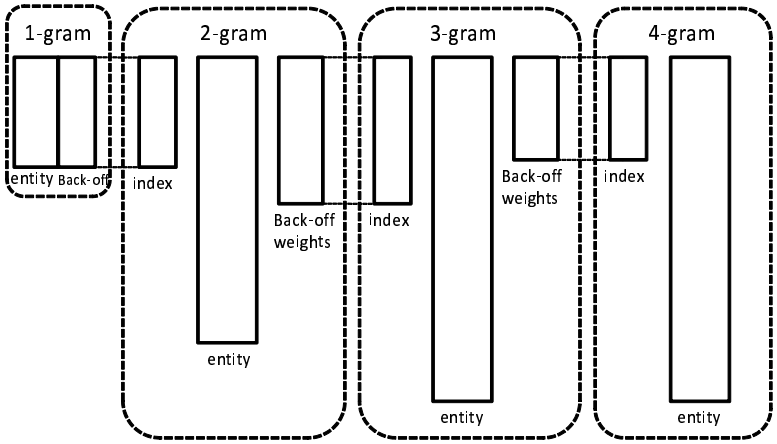
3.4 マルチデコーディング

前節で述べたように，複数モデルを並列に並べて認識できるようになった．特徴量の異なる複数の音響モデルを同時に用いることもできる．また，エンジン動作中の動的なインスタンス無効化・有効化・追加も可能である．

N-gram, 文法，孤立単語認識の 3 つの言語制約を同時に使う場合の Jconf の例を図 2 に示す．マルチデコーディングの指定方法は，通常の Julius のオプションの他に “-AM”, “-LM”, “-SR” を指定する．これらを指定するたび，それ以降が対応する新たなインスタンスの定義となる．それぞれ第一引数としてそのインスタンスの名前を指定する．“-SR” では，名前に続けて使用するインスタンス名を音響・言語の順で指定する．

```
-AM tri -h hmmdefs -hlist logicalTri
-LM word -w wordrecog/station.dict
-LM grammar -gram grammar/price/price
-LM ngram -C dictation/newspaper.jconf
-SR station tri word -b 1000
-SR price tri grammar -looktrellis
-SR newspaper tri ngram -separatescore -b 800 -b2 50
```

図 2: マルチデコーディング用の Jconf 設定の例



$$\mathrm{index}(w_1^{n-1}) = \{w_n \mid C(w_1^n) > \mathrm{cutoff}\}$$
$$\mathrm{entity}(w_1^n) = \log P(w_n \mid w_1^{n-1})$$
$$\mathrm{backoff\_weights}(w_1^n) = \{w_n \mid C(w_1^{n+1}) > \mathrm{cutoff}\}$$

図 3: 単語 N-gram のデータ構造

4 言語制約の拡張

4.1 任意長 N-gram のサポート

N-gram の内部構造を変更し，新たに任意長の N-gram を扱えるようになった．改良された単語 N-gram のデータ構造を図 3 に示す．back-off 係数がつくのは上位の N-gram のコンテキストとなりえるもののみとすることでメモリ量を抑えている[1]，なお，各エントリのインデックスは 24bit で表現されるが，N-gram エントリ数が上限 (約 1677 万個) を越える場合は自動的に 32 ビット表現となる．また，古いバイナリ N-gram を読み込んだ場合は内部で自動的に新しい構造に変換される．

また，従来は 2-gram および後ろ向きの 3-gram の両方が必要であったが，Julius-4 ではこの制約を緩和し，後ろ向き N-gram あるいは前向き N-gram だけでも動作するようになった．どちらか一方だけでも動作するが，従来通り両方を指定することもできる．以下に，指定ごとの Julius-4 の動作を示す．

[1]ただし，ここでは学習時に下位 N-gram のカットオフが上位より厳しくないことを前提としている

```
LOGPROB my_uni(WORD_INFO *winfo, WORD_ID w, LOGPROB
               ngram_prob);
LOGPROB my_bi(WORD_INFO *winfo, WORD_ID context,
              WORD_ID w, LOGPROB ngram_prob)
LOGPROB my_lm(WORD_INFO *winfo, WORD_ID *contexts,
              int clen, WORD_ID w, LOGPROB ngram_prob);
```

図 4: ユーザ定義言語制約関数の型

1. 前向き N-gram のみの場合：第 1 パスではその N-gram 中の 2-gram を使用し，第 2 パスでは，その N-gram からベイズ則に従って確率を逆向きに算出しながら探索する．

$$P(w_1|w_2^N) = \frac{P(w_1, w_2, ..., w_N)}{P(w_2, ..., w_N)} = \frac{\Pi_{i=1}^{N} P(w_i|w_1^{i-1})}{\Pi_{i=2}^{N} P(w_i|w_2^{i-1})}$$

2. 後向き N-gram のみの場合：第 1 パスは，その中の後向き 2-gram から算出した前向き 2-gram 確率を使用し，第 2 パスではそのまま後向き N-gram を適用する．

3. 両方与えられた場合（これまでの Julius と同一）：第 1 パスでは与えられた前向き N-gram 中の 2-gram を使用し，第 2 パスでは与えられた後向き N-gram を適応する．

なお，前向き N-gram だけを用いる場合，第 2 パスで計算途中の文仮説に対して正しい言語スコアが遅れて適用されるため，探索性能が若干低下することに注意が必要である．

また，バイナリ N-gram への変換ツール `mkbingram` も合わせて更新された．上記の全ての組み合わせについてバイナリ N-gram を出力することができる．

4.2 ユーザ定義言語制約

ユーザが定義した言語確率関数を使って認識を行うことができる．図 4 にユーザが定義する関数の形式を示す．それぞれ (1) 単語自身の出現確率，(2) 直前の左単語が与えられたときの単語出現確率（第 1 パス用），(3) 右側の全単語履歴が与えられたときの単語出現確率（第 2 パス用）を表す．これらのユーザ関数がセットされたとき，Julius-4 は認識中の言語スコア計算でその関数を呼び出し，その返り値を言語スコアとして用いて探索を行う．これにより，ユーザが定義する外部言語制約を認識処理過程に直接適用できる．

この機能を使うときは，起動時にオプション “-userlm” を指定する必要がある．さらに N-gram を同時に指定した場合は，上記の関数の引数 `ngram_prob` としてその単語の N-gram 確率が渡されるようになる．これを用いることで，ユーザの与える確率と N-gram 確率の補間なども行える．

4.3 孤立単語認識

単純な音声インタフェースを簡単に構築することを目標として，辞書のみを用いる孤立単語認識の機能を追加した．N-gram や文法と同様，言語モデルのひとつとして扱われる．単語間の近似計算を行う必要がなく，認識は第 1 パスだけで終了できるため，少ない計算量で高速な認識が可能である．

使用するには，単語辞書のみを “-w” オプションで指定する．また，発話開始部および終了部の無音モデルを “-wsil” オプションで指定する．辞書の形式は他の言語モデルと同一である．

第 1 パスのみの認識であるが N ベスト候補を出すことができる． “-output” で出力数を指定する (“-n” ではない)．

4.4 Confusion Network 出力

オプション “-confnet” で認識結果を confusion network [3] の形で出力できるようになった．Julius-4 の内部では，まず単語グラフを従来の方法で算出し，そこから Mangu らの方法 [3] を用いて confusion network を生成する．十分大きなネットワークを生成するには，これまでのグラフ生成と同様，長く探索するように “-n 10” などを付ける必要がある．

4.5 グラフ CM の算出

グラフベースの事後確率計算結果は，”graphcm=...” の形で出力されます．従来の CM もそのまま”cmscore=...” で出力されます．スムージング係数”-cmalpha” は両者で用いられます．

4.6 ショートポーズセグメンテーション

- 動作オプション化： “-spsegment” で使用できるようになった．なお，マルチデコーディング時は，このオプションが付けられた認識処理インスタンスのどれか一つによってセグメンテーションが起こった時点で，全てのインスタンスが同期してセグメンテーションを実行する．
- オプションの追加：ショートポーズと判断する無音モデル名を”-pausemodels sp,sil,silE,silB” のようにリストで指定できるようになった．

5 音声区間検出 (VAD) の強化

5.1 GMM based VAD

GMM に基づく音声区間検出を実装した．使用するには，コンパイル時に`--enable-gmm-vad` をつける．従来の GMM に基づく入力全体の棄却と同様の方法で使用できる．ただし，入力棄却と同時には利用できない．

5.2 デコーダベース VAD

従来のショートポーズセグメンテーションを拡張した，デコーダベースの VAD が行える．これは，認識処理中の仮説状態をもとに有効な音声区間を検出する手法であり，認識用の音響モデル・辞書・言語モデルを用いた判定を行う．[4]．使用するには，コンパイル時に`--enable-decoder-vad` をつけ，実行オプションに “-spsegment” を指定する．

図 5 にデコーダベース VAD の動作の様子を示す．無音区間では，単語仮説を実際には生成しない状態で認識器が動いている．フレームごとの最尤単語が無音単語（無音モデルを読みとする単語，句読点等）でなければ，音声区間開始として処理を通常の認識に切り替える．通常の認識時

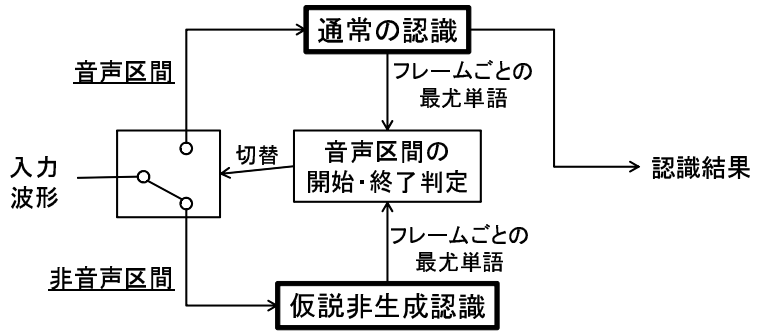


図 5: デコーダベース VAD

表 1: メモリ量削減によるワークエリアのサイズ変化

Julius	MNP-20k	Web-60k	175k-word
3.5.3	50.7 / 13.4	92.4 / 40.4	(> 250)
4.0RC2	47.5 / 10.2	82.0 / 30.0	154.5 / 112.6

値は プロセス全体/ワークエリア，単位は MByte

に，最尤単語が無音単語になれば，そこで音声区間終了として認識を終了し，再び仮説非生成の認識状態に移行する．

マルチデコーディングにおいてデコーダベース VAD を使用する場合，どれかひとつのインスタンスがセグメンテーションの on / off を発見したら全てのインスタンスが同期してセグメンテーションされる仕様になっている．

また GMM 等の棄却機構との併用も可能である．GMM と併用した場合音声区間は AND 判定となる．すなわち，双方の基準が音声になった区間を音声区間とする．

6 その他の変更点

6.1 木構造化辞書のメモリ量削減

木構造化辞書における状態間遷移の内部形式の見直し，および不要なデータ領域の削除を行った．従来は全てのノード間の遷移を linked list で保持していたが，自己遷移と隣のノードについて配列で保持するようにした．起動直後の使用メモリ量の旧バージョンとの比較を表 1 に示す．語彙サイズにもよるが，およそ 25% のワークエリア削減を果たしている．

6.2 仕様変更・バグ修正・ツール追加など

- N-gram で文を自動生成するデバッグツールを追加.
- 48kHz で録音し 16kHz 変換するオプション `-48` の追加.
- adintool の複数 adinnet サーバ対応．また，オプション `-autopause`（一入力ごとに pause），`-loosesync` 複数サーバ間の同期を厳密に行わない）を追加.

- 辞書で第 2 カラムの出力文字列が省略可能になった．省略時は第 1 カラムの内容がそのまま出力される（HTK と同様）
- 辞書の第 1 カラムでクォートが使えるようになった
- jconf 内で環境変数が指定可能になった
- Doxygen で生成されるソースコードドキュメントを更新

6.3 新オプション

- -spsegment
- -pausemodels
- -spmargin
- -spdelay
- -lattice
- -confnet
- -multipath
- -48
- -userlm
- -w dictfile
- -wlist listfile
- -wsil head tail context
- -callbackdebug
- -logfile file
- -nolog
- -AM name
- -LM name
- -SR AMname LMname
- -AM_GMM
- -gmmmargin frames (GMM_VAD)
- -inactive

7　サンプルアプリケーション

Julius-4 には，`julius-simple` というディレクトリに，Julius をライブラリとして組み込んで音声認識を行う C プログラムが同梱されている．

8　おわりに

Julius-4 は，これまでの Julius にない拡張性や柔軟性を目指した，総合的なアップデートとなっている．特に大きな変更点をまとめると以下のようになる．

1. エンジン内部構造の整理とライブラリ化
2. 複数のモデルを用いたマルチデコーディング
3. 言語制約の拡張
4. 音声区間検出 (VAD) の強化
5. メモリの節約

今後は安定化やサンプルアプリの充実，API の拡充等を図るとともに，ユーザどうしの意見交換やノウハウの蓄積の場を作ることを計画している．これらのツールや情報によって，音声認識や音声インタフェース，音声言語処理の研究および応用，実用化の取り組みがさらに促進することを願っている．

参考文献

[1] A. Lee, et al., “An Efficient Two-pass Search Algorithm using Word Trellis Index”, in Proc. ICSLP, pp.1831–1834, 1998.

[2] 大西 翼 他, “ WFST 音声認識デコーダの開発とその性能評価 ”，情報処理学会研究報告，2007-SLP-68, pp.1-6, 2007.

[3] L. Mangu, et al., “Finding consensus in speech recognition: word error minimization and other applications of confusion network,” Computer Speech and Language, vol.14, no.4, pp.373-400, 2000.

[4] 酒井 他, “ 実環境ハンズフリー音声認識のための音響モデルと言語モデルに基づく音声区間検出と認識アルゴリズム ”，電子情報通信学会技術研究報告，SP2007-17，Vol.107, No.116, pp.55–60, 2007.